J Math Imaging Vis (2017) 58:130–146
DOI 10.1007/s10851-016-0694-0

Median Filtering: A New Insight

Sebastián A. Villar[1] · **Sebastián Torcida**[2] · **Gerardo G. Acosta**[1,3]

Received: 19 April 2016 / Accepted: 1 December 2016 / Published online: 27 December 2016

Abstract Median filtering (MF) is a canonical image processing operation truly useful in many practical applications. The MF most appealing feature is its resistance to noise and errors in data, but because the method requires window values to be sorted it is computationally expensive. In this work, a new insight into MF capabilities based on the optimal *breakdown value* (BV) of the median is offered, and it is also shown that the BV-based versions of two of the most popular MF algorithms outperform their corresponding standard versions. A general framework for both the theoretical analysis and comparison of MF algorithms is presented in the process, which will hopefully contribute to a better understanding of the MF many subtle features. The introduced ideas are experimentally tested by using real and synthetic images.

Keywords Image processing · Median filter · Complexity theory · Comparison and evaluation of algorithms · Breakdown value

✉ Sebastián A. Villar
svillar@fio.unicen.edu.ar

Sebastián Torcida
unisebas@gmail.com

Gerardo G. Acosta
g.acosta@uib.es; gerardo.acosta@ieee.org

[1] INTELYMEC (UNCPBA) and CIFICEN (UNCPBA-CICPBA-CONICET), Av del Valle 5737, B7400JWI Olavarría, Argentina

[2] Departamento de Matemática, Facultad de Ciencias Exactas (Campus), UNCPBA, Tandil, Argentina

[3] Grupd' Enginyeria Electrònica (GEE), Departament de Física, Universitat de les Illes Balears, Palma, Spain

1 Introduction

In computer science, algorithmic efficiency refers to the amount of computational resources used by an algorithm. Naturally, to achieve maximum efficiency the usage of resources must be minimized; however, different resources such as time and space cannot be directly compared and the results of analyses depend on how efficiency is measured [1]. Actual efficiency may be in addition affected by implementation choices; the programming language, the way the algorithm is coded, the compiler or compilation options and even the operating system being used can also have their own effect. In summary, the efficiency of different algorithms devised to solve the same problem may differ dramatically. It is acknowledged that differences in design tend to be more significant than differences due to hardware and/or software [2], and because there is no globally unanimous agreement on how algorithmic efficiency should be measured and/or compared, the analysis of algorithmic efficiency is always a major challenge.

In this context, a vast body of the literature has focused on the efficiency of median filtering (MF) algorithms. MF is a sliding-window nonlinear smoothing process, best known for reducing impulsive or salt-and-pepper noise from a digital image while respecting its edges [3]. It is also the foundation upon which more advanced image filters are built [4], and some of its higher level applications include object segmentation, speech and writing recognition, sonar and radar, and others [5–11]. Briefly, the MF window visits each image pixel and places its center on it; all the intensity values within the window are sorted, and the median intensity value is then used to replace the window's center in the filtered image. By using the median value as a consensus, the MF turns out to be more robust and stable than alternative filters.

Probably the main drawback of MF is its computational cost because sorting is a time-consuming task. Many authors have tried to approach this problem; a rather thorough comparison is offered in [12] where (if a squared window of radius size r is assumed as usual) the efficiency of several sorting methods is reviewed: *insertion*, with a $O\left(r^2\right)$ *algorithmic complexity*; *selection*, which is also $O\left(r^2\right)$; *bubble sort*, $O\left(r^2\right)$; *bucket sort*, whose complexity also downs to $O\left(r^2\right)$ when the pixel bit-depth is assumed constant, and more [13–15]. More recently, Tibshirani [16] *binmedian* algorithm used recursive binning and a Chebyshev's approximation of the median by the mean to reach a $O\left(r^2\right)$ complexity in average; alternatively, the interesting idea a double-linked list is introduced by Suomela [17] for MF in 1D, but the corresponding extension to the 2D case is not detailed.

One of the MF main references is Huang et al. algorithm [18] which was the first exhibiting a $O(r)$ complexity. Different approaches have since tried to break this linearity: the Weiss method [19] uses hierarchical histograms to reach a $O(\log r)$ complexity but losing simplicity, and the Gil–Werman method [20] has a $O\left(\log^2 r\right)$ complexity and it is based on trees. A $O(\log r)$ lower bound for the complexity of any two-dimensional MF algorithm was also given in [20]; this claim was refuted by Perreault and Herbert [21], where a variation of Huang et al. algorithm reaching an complexity of $O(b)$ (with $b = 2^{\text{image bit}-\text{depth}}$, the number of scale levels) is presented. In all these cases, complexity is computed *per-pixel* (however, if Huang et al. algorithm were applied to an image of, e.g., *height* by *width* AxB it can be shown that its global complexity would be in fact of $O(AxBxr)$). Among all these algorithms, [18] and [21] are of special interest: the latter claims the lowest complexity to date, while the former is its core.

In this article, a new insight into MF capabilities based on the optimal *breakdown value* (BV) of the median is offered, and BV-based versions of two of the most popular MF algorithms [18,21] are introduced. A general framework to approach the theoretical analysis and comparison of MF algorithms is presented in the process, which uses several abstract and objective metrics that aid in understanding the MF subtle features. All the introduced ideas are experimentally tested by using real and synthetic images.

The structure of this article is the following: in Sect. 2, the main features of Huang et al. and Perreault and Herbert MF algorithms are reviewed. In Sect. 3, a new insight into the MF features is offered based on the optimal BV of the median; this concept is also used to present updated versions of both reviewed algorithms. A rather comprehensive framework for evaluating the theoretical efficiency of algorithms is introduced in Sect. 4: through different metrics the four MF versions at hand are analyzed and compared; experimental results are also given based on actual and synthetic images. The article ends with some final comments in Sect. 5.

2 The MF Algorithm

Let X be the matrix representing a digitized input image of size AxB (*rows by columns*), where $X_{(i,j)}$ denotes the pixel value at the intersection of the image i-th row and the j-th column. The basic scheme is the following: in each step, a squared radius r window (alternative window's shapes could also be used, with the corresponding adaptations [21,22]) is centered at $X_{(i,j)}$; the $(2r+1)^2$ window values are then sorted, their median is computed and then placed in pixel $Y_{(i,j)}$ from the output image Y. The two most popular versions of the MF algorithm are reviewed next.

2.1 Huang et al. Algorithm

The key idea of Huang et al. algorithm (*H_alg*, in the following) is to use a kernel histogram $\boldsymbol{H}$ to store and update all the values from the current window. Four basic steps can be distinguished in the filtering process (the pseudocode is shown in Algorithm A1):

1. *Creating the kernel histogram* $\boldsymbol{H}$. In this step, the data structure to store the kernel histogram $\boldsymbol{H}$ [line 5 in A1(a)] is created, and the memory needed for an array of size $bx1$ is reserved ($b = 2^{\text{image bit}-\text{depth}}$ indicates the number of histogram bins).

2. *Partial initialization (PI) of the kernel histogram* $\boldsymbol{H}$. This step gives $\boldsymbol{H}$ most of its initial values [line 7 in A1(a)], an action to be repeated for every row. It can be further split into two substeps [lines 7.1–7.9 in A1(b)]:

   i. The kernel histogram $\boldsymbol{H}$ is initialized with zero values [lines 7.1–7.3 in A1(b)].
   ii. For every row, the kernel histogram $\boldsymbol{H}$ corresponding to the window centered at the first pixel is partially initialized with the appropriate image values [lines 7.4–7.9 in A1(b)]; this initialization will be completed through the first update step.

3. *Updating the kernel histogram* $\boldsymbol{H}$. Here $\boldsymbol{H}$ is updated while sliding through the image [lines 9–14 in A1(a)]. Figure 1 shows an example for a AxB sized image with a squared window of radius $r = 2$. When the window's center shifts one pixel to the right, e.g., from $X_{(3,3)}$ to $X_{(3,4)}$, the update of $\boldsymbol{H}$ requires those values from $X_{(1,1)}, \ldots, X_{(5,1)}$ to be removed and those values from $X_{(1,6)}, \ldots, X_{(5,6)}$ to be added from $\boldsymbol{H}$; $2r+1$ removals and $2r+1$ additions thus need to be carried out.
4. *Computing the median.* Finally, the median value from the updated kernel histogram $\boldsymbol{H}$ [line 15 in A1(a)] is obtained. Computing the median is done by accumulating frequencies from one extreme of the scale and

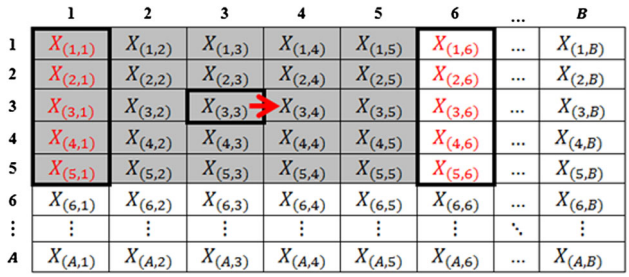

Fig. 1 Each time the window's center shifts to its right, *H_alg* requires $2r+1$ additions to and $2r+1$ removals from the kernel histogram $\boldsymbol{H}$

stopping when the cumulative sum reaches the threshold $\frac{(2r+1)^2+1}{2}$; this is detailed next in Sect. 3 [A4(a)].

(a)

```
1  Input:  Image X of size A x B
           Kernel radius r
           Number of bins b
2  Output: Image Y of the same size as X
3  Aux:    Kernel histogram H of size b x 1
           Auxiliary variable v
4  BEGIN
5    Create kernel histogram H
6    FOR (row ← 1; row <= A; inc +1)
7      PI of H                                  → (b)
8      FOR (col ← 1; col <= B; inc +1)
9        FOR (k ←−r; k <= r; inc +1)
10         v ← X(row+k,col−r−1)
11         H(v) ← H(v) − 1
12         v ← X(row+k,col+r)
13         H(v) ← H(v) + 1
14       END FOR
15       Y(row,col) ← median (H)                → A4(a)
16     END FOR
17   END FOR
18 END
```

(b)

```
7.1  FOR (i ← 0; i < b; inc +1)
7.2    H(i) ← 0
7.3  END FOR
7.4  FOR (i ← −r; i <= r; inc +1)
7.5    FOR (j ← −r; j < r; inc +1)
7.6      v ← X(row+i,j+1)
7.7      H(v) ← H(v) + 1
7.8    END FOR
7.9  END FOR
```

Algorithm A1. Huang et al. algorithm pseudocode (*H_alg*).

Note: for the sake of clarity when describing loops in algorithms, these two cases are distinguished in the following: (i) if a loop index is always used to refer to real elements or structures (e.g., existing image pixels or image column histograms) the corresponding strict range is used; this fact is indicated by an underlined specification of the index; (ii) instead, if a loop index sometimes refers to real elements or structures but at others to artificial ones (e.g., pixel values outside the image limits) all the needed values are included in the corresponding range; in this case, the specification of the index is not underlined.

2.2 Perrault and Hebert's Algorithm

The advantages of the Perreault and Hebert algorithm (*P&H_alg* in the following) over the *H_alg* can be understood if some inefficiencies of the latter are first discussed [21]. In *H_alg* each image pixel value is added to and subtracted from $2r+1$ histograms, and no information is retained between consecutive rows. Intuitively, if each pixel were instead used a constant number of times it could be possible to retain information between rows with the corresponding efficiency increase. To do this, *P&H_alg* uses the additivity property of histograms [19], by which the histogram of the union of two disjoint sets is simply the addition of their respective histograms. The addition of histograms only depends on the number of histogram bins b, itself a function of the image bit-depth.

In this way, the *P&H_alg* reserves one histogram for each image column; this set of histograms is preserved across rows until the whole image is processed (see the pseudocode in Algorithm A2). Essentially, the *P&H_alg* consists of six steps:

1. *Creating the kernel histogram* $\boldsymbol{H}$ *and column histograms* $\boldsymbol{h}^j$ $(1 \leq j \leq B)$. The data structures to store kernel histogram $\boldsymbol{H}$ and column histograms $\boldsymbol{h}^j$ [line 5 in A2(a)] are created. The memory needed to store the $bx1$ bins of $\boldsymbol{H}$ is reserved, just as in A1; in addition, the memory needed to store the B column histograms $\boldsymbol{h}^j$ of size $bx1$ is also reserved.
2. *Partial initialization of column histograms* $\boldsymbol{h}^j$ $(1 \leq j \leq B)$. This step consists of two substeps [lines 6.1 to 6.11 in A2(b)]:
   i. The column histograms $\boldsymbol{h}^j$ are initialized with zero values [lines 6.1–6.5 in A2(b)].
   ii. The values of the first r rows are added to every column histogram $\boldsymbol{h}^j$ $(1 \leq j \leq B)$ [lines 6.6–6.11 in A2(b)]; this initialization will be completed through the first update step.
3. *Initializing the kernel histogram* $\boldsymbol{H}$ *and first update of column histograms* $\boldsymbol{h}^j$ $(1 \leq j \leq B)$. This step is performed on every row of the input image whenever the MF window shifts downward [lines 8.1–8.14 in A2(c)]. Three substeps can be distinguished:
   i. The column histograms $\boldsymbol{h}^j$ $(-r \leq j \leq r)$ are updated by removing and adding the r top and r bottom window pixel values, respectively [lines 8.1–8.6 in A2(c)].

ii. The kernel histogram $\boldsymbol{H}$ is initialized with zero values [lines 8.7–8.9 in A2(c)].
iii. The updated first r column histograms $\boldsymbol{h}^j$ are added to the kernel histogram $\boldsymbol{H}$ [lines 8.10–8.14 in A2(c)]. Histograms $\boldsymbol{H}$ and $\boldsymbol{h}^j$ $(1 \leq j \leq r)$ are afterward ready to be used to compute the median.

4. *Updating the rightmost column histogram* $\boldsymbol{h}^{col+r}$ (where *col* indicates the column of the current window's center)*when the window shifts to the right.* This step is applied every time the MF window shifts to the right [lines 10–13 in A2(a)]. A top and a bottom pixel value are, respectively, removed from and added to the rightmost column histogram $\boldsymbol{h}^{col+r}$.‘
5. *Updating the kernel histogram* $\boldsymbol{H}$. In this step, those values from the window's leftmost column histogram are removed from while those from the new rightmost column histogram are added to the kernel histogram $\boldsymbol{H}$ [lines 14 in A2(a)]; several operations are involved in this task which depend on the image bit-depth but not on the radius size r, as in *H_alg* [lines 14.1–14.3 in A2(a)]. Worth mentioning here that Perreault and Herbert use a parallel processing *SIMD* (*Single Instruction Multiple Data*) hardware optimization to improve runtime performance; this fact was also noted by *Alekseychuk* [23], who attempted to lower the runtime constant but could not show practical improvements.
6. *Computing the median.* This step is analogous to that from *H_alg*, [see line 15 in A1(a) and A2(a)], which is detailed next in Sect. 3 [A4(a)].

3 A New Insight Into the MF Algorithm

It is acknowledged that as a summary measure, the median is much more stable and hence highly more reliable than the classical mean or average [24]. In fact, the mean of a set of numbers changes when at least one of those numbers is changed or replaced; the median, instead, does not necessarily change even if almost 50% of the numbers are changed or replaced. E.g., assume that n (odd) ordered numbers are at hand; if the $\frac{(n+1)}{2} - 1$ lowest numbers are replaced by arbitrary other numbers but keeping them below the $\frac{(n+1)}{2}$ ranked number (the median), the new median will remain the same.

(a)

```
1  Input:  Image X of size A x B
           Kernel radius r
           Number of bins b
2  Output: Image Y of the same size as X
3  Aux:    Kernel histogram H of size b x 1
           Column histograms h^j (1 ≤ j ≤ B) of size b x 1
           Auxiliary variable v
4  BEGIN
5    Create histograms H and h^j
6    PI of h^j                                   → (b)
7    FOR (row ← 1; row <= A; inc +1)
8      PI of H and 1st update of h^j             → (c)
9      FOR (col ← 1; col <= B; inc +1)
10       v ← X(row-r-1,col+r)
11       h^col+r(v) ← h^col+r(v)-1
12       v ← X(row+r,col+r)
13       h^col+r(v) ← h^col+r(v)+1
14       H(·) ← H(·) + h^col+r(·) − h^col-r-1(·)   → 14.1 FOR (i ← 0; i < b; inc +1)
                                                     14.2   H(i) ← H(i) + h^col+r(i) − h^col-r-1(i)
                                                     14.3 END FOR
15       Y(row,col) ← median (H)                   → A4(a)
16     END FOR
17   END FOR
18 END
```

(b)

```
6.1  FOR (i ← 0; i < b; inc +1)
6.2    FOR (col ← 1; col <= B; inc +1)
6.3      h^col(i)← 0
6.4    END FOR
6.5  END FOR
6.6  FOR (j ← 1; j <= r; inc +1)
6.7    FOR (col ← 1; col <= B; inc +1)
6.8      v ← X(j,col)
6.9      h^col(v) ← h^col(v) + 1
6.10   END FOR
6.11 END FOR
```

(c)

```
8.1  FOR (j ← −r; j <= r; inc +1)
8.2    v ← X(row-r-1,j)
8.3    h^j(v) ← h^j(v) − 1
8.4    v ← X(row+r,j)
8.5    h^j(v) ← h^j(v) + 1
8.6  END FOR
8.7  FOR (i ← 0; i < b; inc +1)
8.8    H(i) ← 0
8.9  END FOR
8.10 FOR (i ← 0; i < b; inc +1)
8.11   FOR (j ← 1; j <= r; inc +1)
8.12     H(i) ← H(i) + h^j(i)
8.13   END FOR
8.14 END FOR
```

Algorithm A2. Perreault and Hebert algorithm pseudocode (*P&H_alg*).

This property can be described through the concept of *breakdown value*: in general terms, it denotes the percentage of data in a set that could be arbitrarily replaced without grossly modifying the value of an estimation or a computation. Clearly, the mean has a 0% BV while the median has almost a 50% BV [24].

Each time the MF window shifts some of its values are removed and simultaneously replaced by new values, and a new median is computed. Two shortcomings can be identified in this task: first, the median computation typically uses a *bottom-up* accumulating strategy (e.g., for an 8-bit grayscale where values range from 0 to 255, frequencies are accumulated from the zero value and upward) until the $\frac{(2r+1)^2+1}{2}$ ranked value is reached; in this way, if the MF window eventually processes an image region with most values near the top of the grayscale, the algorithm will get slower (an analogous problem would take place if frequencies were accumulated *top-down* and image regions with most of low pixel values were eventually found). Second: it seems rather inefficient not to take into account that successive windows share most of their values and thus result in similar medians. More precisely: the proportion of shared values between consecutive radius r squared windows is essentially $\frac{(2r+1)^2-2(2r+1)}{(2r+1)^2}$ (the exception being the case when the window shifts downward, assuming the typical left-right and top-down image processing). The percentage of shared information between successive windows thus increases really fast with the radius: a 33% of shared values for a radius $r = 1$; a 60% of shared values for a radius $r = 2$, a 90% of shared values for a radius $r = 10$ and so on. Both mentioned inefficiencies can be overcome by making the most of the median optimal BV: the median from a new window can be computed significantly faster by retaining the median from the previously processed window and updating it; in turn, this strategy enables a more efficient processing of those image regions with values in any extreme of the scale handling equally both cases. It is experimentally verified in Sect. 4 that this approach leads to a significant increase in efficiency as the bit-depth, the radius size and even the number of channels increases. Based on these considerations, updated formulations of both *H_alg* and *P&H_alg* are offered next.

3.1 BV-Based Version of H_alg

The *H_alg* (A1) is now revisited to incorporate the BV concept either by adding or modifying some of its instructions; the corresponding changes are light colored in Algorithm A3.

(a)

```
1  Input: Image X of size A x B
          Kernel radius r
          Number of bins b
2  Output: Image Y of the same size as X
3  Aux: Kernel histogram H of size b x 1
       Previous median variable Pm
       Lower than previous median variable Lm
       Greater than previous median variable Gm
       Auxiliary variable v
4  BEGIN
5    Create kernel histogram H
6    FOR (row ← 1; row <= A; inc +1)
7      PI of H; init.Pm, Lm, Gm  ──→ (b)
8      FOR (col ← 1; col <= B; inc +1)
9        FOR (k ←−r; k <= r; inc +1)
10         v ← X(row+k,col−r−1)
11         H(v) ← H(v) − 1
12         IF (v ≥ Pm) THEN
13           Gm ← Gm - 1
14         ELSE
15           Lm ← Lm - 1
16         END IF
17         v ← X(row+k,col+r)
18         H(v) ← H(v) + 1
19         IF (v ≥ Pm) THEN
20           Gm ← Gm + 1
21         ELSE
22           Lm ← Lm + 1
23         END IF                    ──→ A4 (b)
24       END FOR
25       Y(row,col) ← median (H, Pm, Lm, Gm)
26     END FOR
27   END FOR
28 END
```

(b)

```
7.1  FOR (i ← 0; i < b; inc +1)
7.2    H(i) ← 0
7.3  END FOR
7.4  Pm ← 0, Lm ← 0, Gm ← 0
7.5  FOR (i ← −r; i <= r; inc +1)
7.6    FOR (j ← −r; j < r; inc +1)
7.7      v ← X(row+i,j+1)
7.8      H(v) ← H(v) + 1
7.9      IF (v ≥ Pm) THEN
7.10       Gm ← Gm + 1
7.11     ELSE
7.12       Lm ← Lm + 1
7.13     END IF
7.14   END FOR
7.15 END FOR
```

Algorithm A3. Pseudocode of the *H_alg* BV-based version.

Three main modifications can be distinguished:

1. *Initializing the auxiliary variables Pm, Lm and Gm.* Through this step [line 7 in A3(a)], the kernel histogram $\boldsymbol{H}$ is partially initialized and the new auxiliary variables *previous median Pm*, *lower than (previous) median Lm* and *greater or equal than (previous) median Gm* are all initialized; this must be done for every image row. The

initialization of $\boldsymbol{H}$ is just the same as in A1(b), while the initialization of variables Pm, Lm and Gm consists of two substeps:

i. Their values are set to zero [line 7.4 in A3(b)].
ii. Before its addition to $\boldsymbol{H}$, each new pixel value is compared with the previous median Pm: if the new value is greater or equal than Pm, Gm is incremented in one and otherwise Lm is incremented in one otherwise [lines 7.9–7.13 in A3(b)]. Whenever a new row is to be processed, variables Pm, Gm and Lm have all to be initialized with 0; a more efficient procedure is again to retain their values from the previous processed row.

2. *Updating the auxiliary variables Lm and Gm*. Each time a pixel value is either added to or removed from the kernel histogram $\boldsymbol{H}$ after being compared with Pm, variables Lm and Gm are correspondingly updated [lines 12–16 and 19–23 in A3(a)].
3. *Computing the new median.* This step computes the new window's median by using the kernel histogram $\boldsymbol{H}$ and the auxiliary variables Lm, Gm and Pm [line 25 in A3(a)]. For a better understanding of this step, an example is given next which compares the standard median computation [A4(a)] and the BV-based one [A4(b)].

In A4(a) the median is computed by accumulating frequencies in $\boldsymbol{H}$ until a specific threshold T is reached [lines 7–10 in A4(a)]; the value of T naturally depends on the window's radius r, but also on the window's shape and on the distribution of the updated pixel values in $\boldsymbol{H}$. For a squared window of radius r is $T = \frac{(2r+1)^2+1}{2}$ for a complete window, and this value decreases when incomplete windows are found at the image edges. Algorithm A4(b) shows instead the BV-based computation of median. First, the previous median Pm is stored in the auxiliary variable m [line 5 in A4(b)]; next, the threshold T is computed [line 6 in A4(b)] and compared with Lm [line 7 in A4(b)]: if $T \leq Lm$, the new median will be lower than the previous one and it will be found moving downward from the current Pm bin in the kernel histogram $\boldsymbol{H}$ [lines 8–14 in A4(b)]; otherwise, the new median will be greater than the previous one and will be found moving upward from the current Pm bin in $\boldsymbol{H}$ [lines 16–21 in A4(b)]. The auxiliary variables Lm and Gm are accordingly updated in the process [lines 11–13 or 18–20 in A4(b)].

Consider now this situation: a squared window W of radius $r = 1$ with $(2r+1)^2 = 9$ pixel values is given; then $T = \left[\frac{(2r+1)^2+1}{2}\right] = 5$. Assume that $W = [1, 3, 3, 4, 5, 5, 7, 7, 9]$ are current window's ordered values, so $Pm = 5$, $Lm = 4$ and $Gm = 5$. Assume next that (the right new column) values 1, 1 and 1 are added to W while (the leftmost column) values 7, 7 and 1 are removed from it. This update of W results in $W = [1, 1, 1, 3, 3, 4, 5, 5, 9]$ which in turn updates $Lm = 6$ and $Gm = 3$ following the comparison with $Pm = 5$. Since $T = 5 \leq 6 = Lm$, the new median will be found moving downward from the $Pm = 5$ bin in the kernel histogram $\boldsymbol{H}$. Here, only two bin-displacements through $\boldsymbol{H}$ are needed to obtain the new median value; the auxiliary variables are accordingly updated to $Pm = 3$, $Lm = 3$ and $Gm = 6$, respectively. Since T is now greater than Lm, the displacement will next start from Pm and upward.

(a) Standard

```
1  Input:  Kernel histogram H of size b x 1
2  Output: Median variable m
3  Aux:    Threshold variable T
           Accumulator variable acc
4  BEGIN
5    acc ← 0
6    T ← ((2r+1)^2+1)/2
7    FOR (i←0; (i< b) AND (acc < T); inc +1)
8      acc ← acc + H(i)
9      m ← i
10   END FOR
11   Return ← m
12 END
```

(b) BV-based

```
1  Input: Kernel histogram H of size b x 1
          Previous median variable Pm
          Lower than previous median Lm
          Greater than previous median Gm
2  Output: Previous median variable Pm
3  Aux:    Threshold variable T
           Auxiliary variable v, m
4  BEGIN
5    m ← Pm
6    T ← ((2r+1)^2+1)/2
7    IF (T ≤ Lm) THEN
8      v ← H(m)
9      FOR (i←m; (i>=0) AND (T ≤ (Lm-v)); inc -1)
10       Pm ← i − 1
11       v ← H(i)
12       Lm ← Lm - v
13       Gm ← Gm + v
14     END FOR
15   ELSE
16     FOR (i←m + 1; (i<b) AND (T > Lm); inc +1)
17       Pm ← i
18       v ← H(i)
19       Lm ← Lm + v
20       Gm ← Gm - v
21     END FOR
22   END IF
23   Return ← Pm
24 END
```

Algorithm A4. Computing the median.

3.2 BV-Based Version of the P&H_alg

The *P&H_alg* A2 is now revisited to incorporate the BV concept by either adding or modifying some instructions which are light colored in Algorithm A5.

(a)

```
1  Input: Image X of size A x B
          Kernel radius r
          Number of bins b
2  Output: Image Y of the same size as X
3  Aux: Kernel histogram H of size b x 1
       Column histograms h^j (1 ≤ j ≤ B) of size b x 1
       Previous median variable Pm
       Lower than prev. med. variable Lm
       Greater than prev. med. variable Gm
       Auxiliary variables v, v1, v2
4  BEGIN
5    Create histograms H and h^j
6    PI of h^j                                   A2(b)
7    FOR (row ← 1; row <= A; inc +1)
8      PI H; Init. Pm, Lm and Gm; 1st update of h^j
9      FOR (col ← 1; col <= B; inc +1)
10       v ← X(row-r-1,col+r)
11       h^(col+r)(v) ← h^(col+r)(v) − 1
12       v ← X(row+r,col+r)
13       h^(col+r)(v) ← h^(col+r)(v) + 1
14       FOR (i ← 0; i < b; inc +1)
15         v1 ← h^(col+r)(i)
16         v2 ← h^(col-r-1)(i)
17         H(i) ← H(i) + v1 − v2
18         IF (i ≥ Pm) THEN
19           Gm ← Gm + v1
20         ELSE
21           Lm ← Lm + v1
22         END IF
23         IF (i ≥ Pm) THEN
24           Gm ← Gm - v2
25         ELSE
26           Lm ← Lm - v2
27         END IF
28       END FOR
29       Y(row,col) ← median (H, Pm, Lm, Gm)
30     END FOR
31   END FOR
32 END
```

(b)

```
8.1  FOR (j ← −r; j <= r; inc +1)
8.2    v ← X(row-r-1,j)
8.3    h^j(v) ← h^j(v) − 1
8.4    v ← X(row+r,j)
8.5    h^j(v) ← h^j(v) + 1
8.6  END FOR
8.7  FOR (i ← 0; i < b; inc +1)
8.8    H(i) ← 0
8.9  END FOR
8.10 Pm ← 0, Lm ← 0, Gm ← 0
8.11 FOR (i ← 0; i < b; inc +1)
8.12   FOR (j ← 1; j <= r; inc +1)
8.13     H(i) ← H(i) + h^j(i)
8.14     IF (i > Pm) THEN
8.15       Gm ← Gm + h^j(i)
8.16     ELSE
8.17       Lm ← Lm + h^j(i)
8.18     END IF
8.19   END FOR
8.20 END FOR
```

Algorithm A5. BV-based version of *P&H_alg* pseudocode.

Three main changes can be distinguished:

1. *Partial initialization of the kernel histogram* $\boldsymbol{H}$ *and initialization of auxiliary variables* Pm, Lm *and* Gm. This step is similar to step 3 from A2(c). Only two modifications are made [see A5(b)]:
   i. The auxiliary variables Pm, Lm and Gm are all initialized with zero values [line 8.10 in A5(b)].
   ii. Each pixel value in the window is compared with the previous median Pm: if its value is greater or equal Pm, variable Gm is incremented in one; otherwise, variable Lm is incremented in one [lines 8.14–8.18 in A5(b)].
2. *Updating the auxiliary variables* Lm *and* Gm. This is analogous to step 2 in Sect. 3.1 [lines 15–27 in A5(a)].
3. *Computing the new median.* This is analogous to step 3 in Sect. 3.1 [line 29 in A5(a)].

4 A Theoretical Framework for MF Analysis and Comparison

A general framework for the analysis and comparison of MF algorithms at the theoretical level is now presented, which is used in the process to evaluate the four discussed versions. As mentioned, algorithmic efficiency is an important part of the computational complexity theory which aims to provide estimates of the resources needed by any algorithm solving a given computational problem, such estimates provide valuable insight to design more efficient algorithms.

The performance of algorithms can be studied independently of specific languages or machines, and several tools aid in assessing their efficiency without considering implementation. Among them, the widely known asymptotic analysis of worst-case complexity [1,2] and the *random access machine* abstract model of computation are indeed relevant. Generally speaking, the latter is a rather simple theoretical model which helps to understand how an algorithm performs on actual machines; although alternative theoretical models exist [25,26], the random access machine model strikes a fine balance between capturing the essential behavior of the computer and being simple to work with. In addition, it has proven useful in practice: under this model, the algorithmic analysis of efficiency seldom produces substantially misleading results. The framework presented next is based on all these considerations.

4.1 Theoretical Framework

Table 1 exhibits the comparison for the two analyzed algorithms both in their standard and BV-based versions. The MF was applied to an input image X of size $A \times B$ by using a squared window of radius r that produced a filtered output image Y of the same size.

To be consistent with the theoretical model described above, this comparison should ideally be based on metrics

Table 1 Comparison metrics and the corresponding results for the four analyzed MF algorithms

Algorithm	AC	DM	SM	Operation	Type case	DMA	AO	LC	TE
H_alg	$O(r)$	X of size AxB Y of size AxB $\boldsymbol{H}$ of size $bx1$	r, b, v, acc, T, m	Histogram update	Worst case	$6(2r+1)+1$	$8(2r+1)$	$(2r+1)$	b
					Average case				
					Best case				
				Median computation	Worst case	b	$2b+5$	$2b$	
					Average case	$\frac{b}{2}$	$b+5$	b	
					Best case	1	7	2	
BV-based H_alg			$r, b, v, T, m, Pm, Lm, Gm,$	Histogram update	Worst case	$6(2r+1)+1$	$10(2r+1)$	$3(2r+1)$	2
					Average case				
					Best case				
				Median computation	Worst case	b	$5b+5$	$2b+1$	
					Average case	$\frac{b}{2}$	$\frac{5b}{2}+5$	$b+1$	
					Best case	1	10	3	
P&H_alg	$O(b)$	X of size AxB Y of size AxB $\boldsymbol{H}$ of size $bx1$ $\boldsymbol{h}^j (1 \le j \le B)$ of size $bx1$	r, b, v, acc, T, m	Histogram update	Worst case	$4b+7$	$6b+11$	b	b
					Average case				
					Best case				
				Median computation	Worst case	b	$2b+5$	$2b$	
					Average case	$\frac{b}{2}$	$b+5$	b	
					Best case	1	7	2	
BV-based P&H_alg			$r, b, v, T, m, Pm, Lm, Gm, v_1, v_2$	Histogram update	Worst case	$4b+7$	$8b+11$	$3b$	2
					Average case				
					Best case				
				Median computation	Worst case	b	$5b+5$	$2b+1$	
					Average case	$\frac{b}{2}$	$\frac{5b}{2}+5$	$b+1$	
					Best case	1	10	3	

that are independent from specific languages or machine implementations [25,26]. The chosen ones are capable of capturing differences in the theoretical efficiency of the algorithms; some of them are space metrics, while the remaining ones are time metrics:

- *algorithmic complexity* (AC);
- *dynamic memory* (DM);
- *static memory* (SM);
- *dynamic memory accesses* (DMA);
- *arithmetic operations* (AO)
- *logic comparisons* (LC).

A last ad hoc time metric to specifically measure the efficiency of the median computation was also included: the *transition effort* (TE), interpreted as the effort rate associated with the transition between consecutive median values in the MF dynamic, emerged as a useful tool to explain the BV-based algorithms superior experimental performance.

For most of these metrics the worst, the average and the best case (defined, respectively, by the maximum, the average and the minimum number of steps taken in any instance) were considered, but only the worst case was used for the introduced ad hoc metric. Metrics DM, SM and TE are globally quantified while AC, DMA, AO and LC are quantified per-pixel, as usual.

As shown in Table 1, the AC for both the standard and the BV-based versions of the *H_alg* (A1 and A3, respectively) is simply $O(r)$: i.e., the AC does not vary with the median computation method. A similar pattern is found when comparing the standard *P&H_alg* (A2) with the BV-based *P&H_alg* (A5): both share an AC of $O(b)$.

To compute the DM for each of the four MF algorithms (A1-A3 and A5), the amount of memory needed to allocate both the input and output images of size AxB has to be established and also the corresponding amount of memory to store the kernel histogram $\boldsymbol{H}$ of size $bx1$. MF algorithms A3 and A5 need extra DM due to the B column histograms $\boldsymbol{h}^j$ $(1 \leq j \leq B)$. On the other hand, the BV-based MF algorithms A2 and A5 need more SM involved in managing variables Pm, Gm, Lm and the remaining auxiliary ones.

For each pixel value, the *H_alg* requires [lines 9–15 in A1(a)]:

i. A loop of six instructions to update $\boldsymbol{H}$. This loop needs $2r+1$LC and $2r+1$ AO [used by the loop index k, where $-r \leq k \leq r$; line 9 in A1(a)]. Inside the loop, the addition and the removal of values [lines 10–13 in A1(a)] require six DMA (to read the pixel value from image X, to read the previous value from kernel histogram $\boldsymbol{H}$ and to update the new value in $\boldsymbol{H}$) and seven AO (to compute the indexes from image X and for the addition/substraction operations) are needed. In total $6(2r+1)$ DMA, $8(2r+1)$ AO and $2r+1$ LC, respectively.
ii. A function to calculate the median value [line 15 in A1(a) which uses A4(a)]. The initialization of the threshold T requires five AO [line 6 in A4(a)] and a loop with four instructions [lines 7–10 in A4(a)] is also needed to compute the median. In general, the worst case needs $2b$ LC (since $0 \leq i < b$ and $acc < T$), b DMA (to read the pixel value from $\boldsymbol{H}$) and $2b$ AO (b increments of the loop index i, and b additions to accumulate the kernel histogram values); the average case uses b LC, $\frac{b}{2}$ DMA and b AO, while the best case uses two LC, one DMA and two AO. Globally, the TE worst case for the standard computation of the median is obtained as the rate of maximum effort transitions between consecutive median values: the maximum effort takes place each time the new median is exactly b when transitioning from any of the b possible current median values. Therefore, b different maximum effort transitions are associated with the worst case, an occurrence rate that varies linearly with b.
iii. Finally, one DMA is required to store the median in the output image Y [line 15 in A1(a)].

The BV-based *H_alg* [A3(a)] requires the same amount of DMA but more AO and more LC than the standard *H_alg* [A1(a)]:

i. Two LC and two AO are required to update the auxiliary variables Lm and Gm [lines 12–16 and lines 19–23 in A3(a)]. This results in $6(2r+1)$ DMA, $10(2r+1)$ AO and $3(2r+1)$ LC, respectively.
ii. An additional function uses the BV concept to compute the median [line 25 in A3(a), using A4(b)]. The initialization of the threshold T requires the same five AO as in the standard version [line 6 in A4(b)]. In addition, one LC is used to establish the direction of search [line 7 in A4(b)]. If the condition $T \leq Lm$ is true, a loop of seven instructions to obtain the median value [lines 8–14 in A4(b)] is needed; more precisely, the worst case requires $2b$ LC (since $0 \leq i < b$ and $T \leq Lm - \boldsymbol{H}(i)$), b DMA (to read the pixel value from $\boldsymbol{H}$) and $5b$ AO (b subtractions to check $T \leq Lm - \boldsymbol{H}(i)$, b increments/decrements for the loop index i and $3b$ to update variables Pm, Lm and Gm). If the condition $T \leq Lm$ is instead false, a loop of five instructions to obtain the median value [lines 16–21 in A4(b)] is required; in this case, the same amount of DMA and LC resources are required as before, but a fewer AO. In summary, the worst case requires b DMA, $5b$ AO and $2b$ LC; for the average case $\frac{b}{2}$ DMA, $\frac{5b}{2}$ AO and b LC are used, while for the best case one DMA, five AO and two LC are needed. The TE worst case for the BV-based computation of the median is obtained as the rate

of maximum effort transition between consecutive medians: this maximum effort is b, which takes place when transitioning from a current median 0 to a new median b or vice versa. Only two realizations of the worst-case result in a TE rate that is therefore constant.

iii. Finally, one DMA is required to store the median result [line 26 in A3(a)].

The amount of needed resources for both the standard and the BV-based *H_alg* is shown in Table 1.

For each pixel, the *P&H_alg* requires [lines 10–15 in A2(a)]:

i. Five instructions to update the column histograms $\boldsymbol{h}^j$ ($1 \leq j \leq B$) and the kernel histogram $\boldsymbol{H}$. Removing or adding a pixel value from $\boldsymbol{h}^j$ [lines 10–11 or 12–13 in A2(a), respectively] requires six DMA (to read the pixel value from image X, to read the previous value from column histogram $\boldsymbol{h}^j$ and to update the new value in $\boldsymbol{h}^j$) and eleven AO (to compute indexes from locations at image X and at column histograms $\boldsymbol{h}^j$, and also for the substraction/addition operations). A loop of three instructions to update the kernel histogram $\boldsymbol{H}$ [lines 14.1–14.3 in A2(a)] requires $4b$ DMA ($b+b=2b$ readings used in the addition/removal of values from $\boldsymbol{h}^j$, b readings of the current $\boldsymbol{H}$ values and b updates in $\boldsymbol{H}$), $6b$ AO (to compute indexes from column histograms $\boldsymbol{h}^j$ and an extra addition/removal operation) and b LC (for the loop index i, where $0 \leq i < b$).

ii. The same function as in *H_alg* [A4(a)] to compute the median value. The initialization of the threshold T requires five AO [line 6 in A4(a)], and a loop of four instructions [lines 7 to 10 in A4(a)] enables to compute the median. Typically, the worst case needs $2b$ LC, b DMA and $2b$ AO; the average case in turn needs b LC, $\frac{b}{2}$ DMA and b AO, while the best case needs two LC, one DMA and two AO. Finally, one DMA is required to assign the median value in the output image Y [line 15 in A2(a)]. The TE worst case linearly depends on b, just as in the standard *H_alg*.

The BV-based *P&H_alg* [A5(a)] requires the same amount of DMA, but more AO and LC resources than the standard *P&H_alg* [A2(a)]:

i. $2b$ LC and $2b$ AO are required to update variables *Lm* and *Gm* [lines 18–27 in A5(a)].

ii. Using the BV, the median value is computed by using the same function as in the standard *P&H_alg* [A4(b)]. Initializing the threshold T requires five AO [line 6 in A4(b)]. In addition, one LC is needed to establish the direction of search [line 7 in A4(b)] and a seven/six instructions [lines 8–14 or lines 16–21 in A4(b)] are needed to compute the median. In summary, the worst case requires b DMA, $5b$ AO and $2b$ LC; for the average case $\frac{b}{2}$ DMA, $\frac{5b}{2}$ AO and b LC are used, while for the best case one DMA, five AO and two LC are needed. The TE worst case is constant, just as in the BV-based *H_alg*.

iii. Finally, one DMA is required to store the median result [line 29 in A5(a)].

The amount of needed resources for both the standard and the BV-based *P&H_alg* is shown in Table 1.

In summary, in terms of DM both the standard and the BV-based *P&H_alg* require more resources than their *H_algm* analogues due to the use of column histograms $\boldsymbol{h}^j$ ($1 \leq j \leq B$). On the other hand, both BV-based MF algorithms require more SM, more LC and more AO resources either in the worst, the average and the best case. This fact is balanced by the highest efficiency of the BV-based versions in terms of the ad hoc time metric TE.

4.2 Experimental Results

To experimentally test the theoretical results, a comparison based on common implementations for each of the four considered MF versions is required; a specific environment was therefore defined to instantiate all of them. They were first implemented in MATLAB® and afterward migrated to an Integrated Development Environment (IDE) QtCreator from Nokia for GNU/Linux in C++ code. Tests were performed in an Intel® Core™i7-3630QM with 8-GB RAM 2.4GHz CPU, and the chosen operating system was Ubuntu 14.04 LTS (32 bits). It should be apparent that alternative implementations in other programming languages and/or actual machines are also possible; for some of these languages and machines, specific optimization shortcuts can be found but they highly depend on the implementation strategy (such as SIMD instructions [21,27], cache friendliness [21], multilevel histograms [28] and others).

Figure 2 exhibits the experimental results processing time against radius size for the common implementation of the four MF versions; only in this context the processing time makes sense as a measure of experimental efficiency. These four MF versions were tested on the classic side-scan sonar image of a sunken boat from Lake Washington[1] (Fig. 3, center), whose size is 470×470 (AxB) pixels from an 8 bit ($b = 256$) grayscale. In Fig. 2, the vertical axis indicates the processing time (in seconds) while the horizontal axis indicates the MF window's radius size.

As shown, the processing time varies linearly with the radius size for the *H_alg* versions; conversely, the *P&H_alg*

[1] Available: http://www.edgetech.com/.

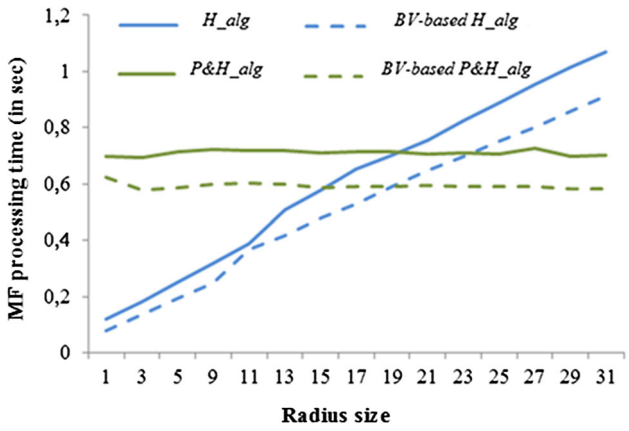

Fig. 2 *Processing time* versus *radius size* for the four MF versions

versions processing time does not depend on the radius size (e.g., for a window's radius of up to $r = 20$, consisting of $(2r + 1)^2 = 1681$ window pixel values, the *H_alg* is faster than the *P&H_alg* in both versions). It is also shown that each MF algorithm (*H_alg* and*P&H_alg*) exhibits approximately the same trend for both versions, but lower processing times are consumed by the BV-based ones. Worth noting that the extra computational resources required by the corresponding BV-based versions (e.g., in terms of SM, AO or LC) do not result in additional processing time; this feature could not be perceived if only the AC metric were used for the comparison.

In Fig. 3, five different *regions of interest* (ROI) extracted from the test image are shown; their sizes are 100x100 (A x B) pixels in every case, and the cartesian coordinates of the

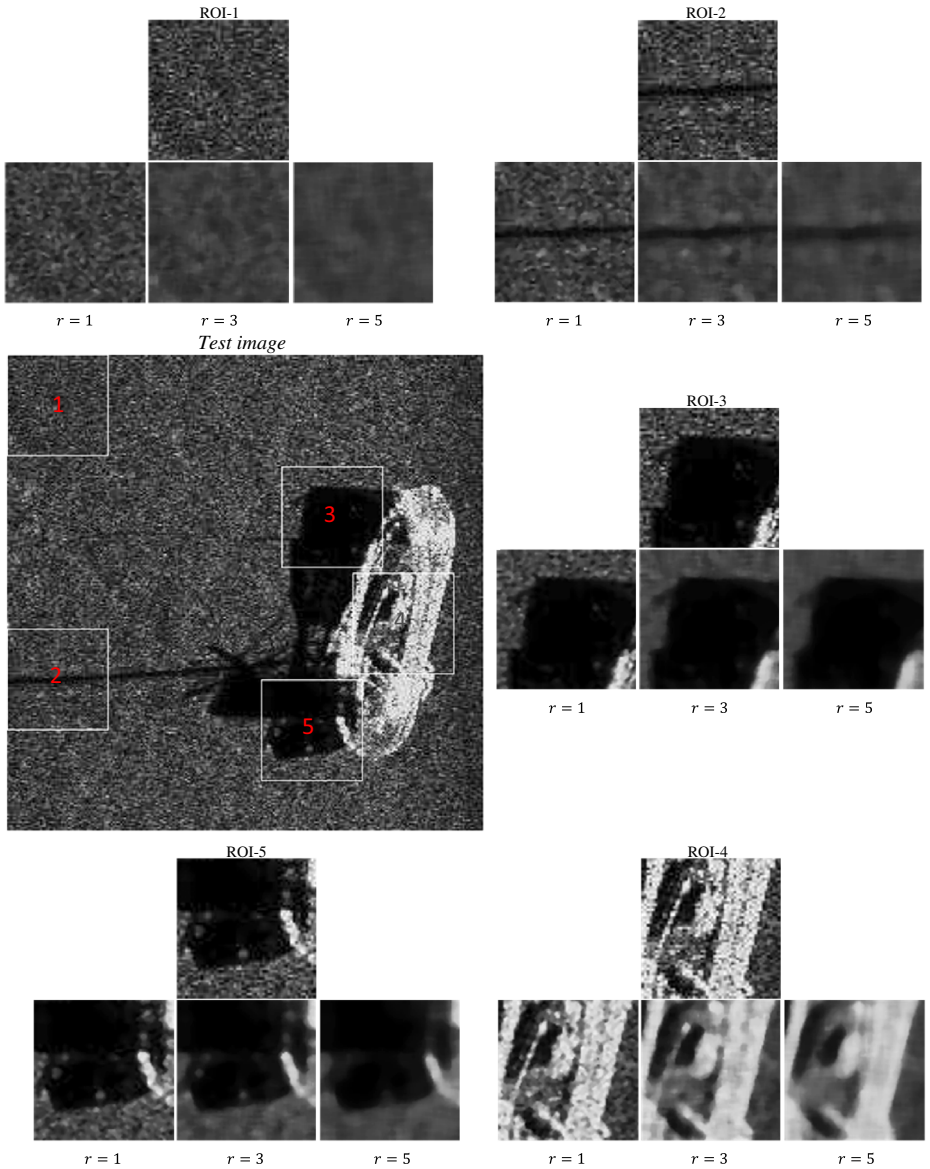

Fig. 3 Test image: sunken boat in Lake Washington (center). Five ROI and the corresponding MF outputs for radius sizes $r = 1, 3, 5$

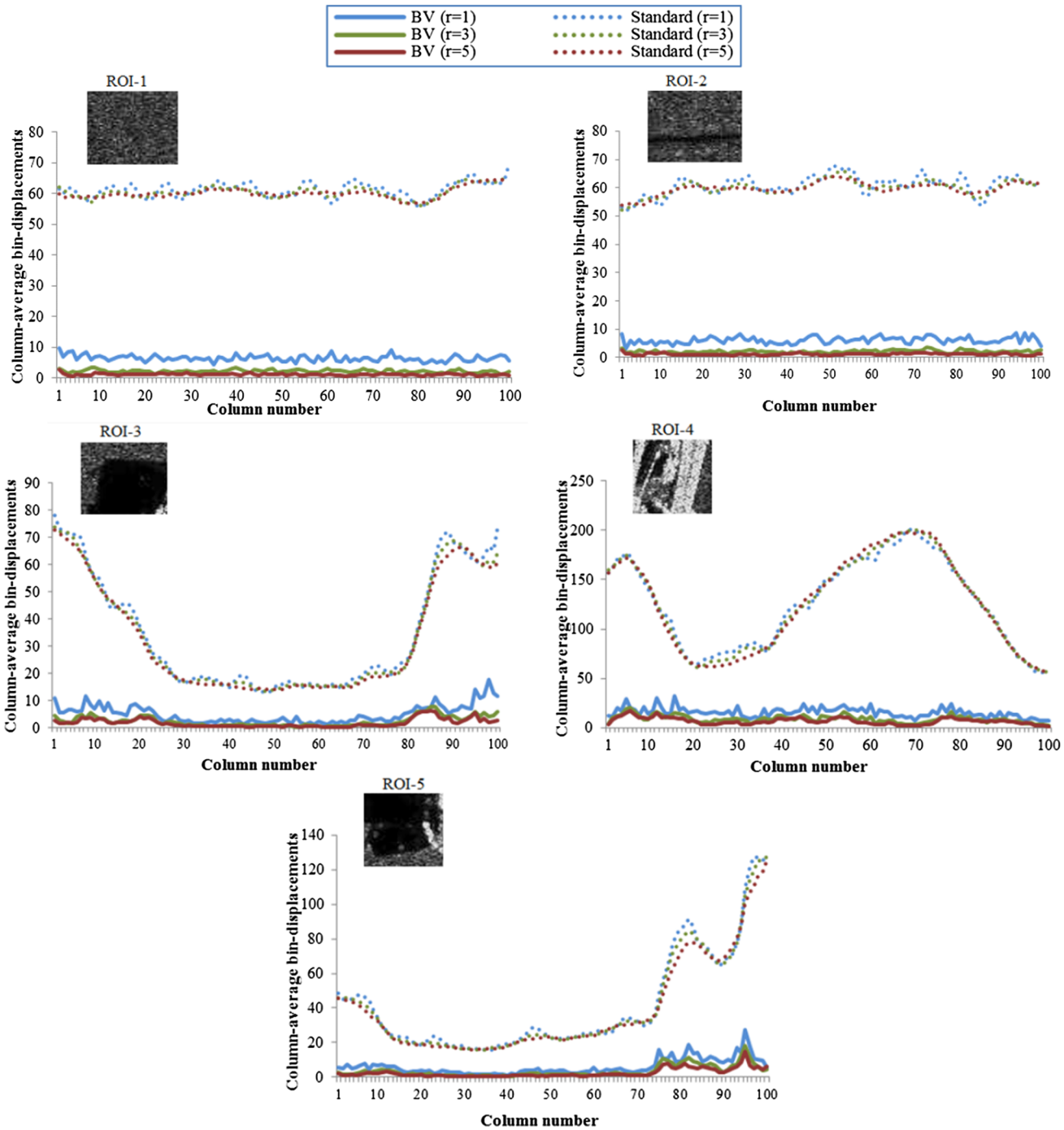

Fig. 4 *Column-average bin-displacements* versus *column number* for each ROI when MF radius sizes $r = 1, 3, 5$ are used

corresponding upper-left vertex used as reference are, respectively: (i) (0, 0); (ii) (270, 0); (iii) (110, 270); (iv) (215, 340) and (v) (320, 250). These five sampled regions were deliberately chosen to capture different features often found in side-scan sonar image processing [10,29,30]: (1) seabed reverberation; (2) seabed reverberation and little acoustic shadow; (3) seabed reverberation and acoustic highlight plus a significant shadow zone; (4) the highest acoustic highlight, and (5) seabed reverberation and shadow zones plus strong acoustic highlight.

Figure 3 exhibits the corresponding filtered images when radius sizes $r = 1$ (9 processed pixels in every window), $r = 3$ (49 processed pixels) and $r = 5$ (121 processed pixels) are used. The smaller the radius size, the smaller number of processed pixel values in each iteration of the MF; the output image will include more geometrical details but also will be

Table 2 Min–max column-average bin-displacements comparison by radius size

	$r=1$				$r=3$				$r=5$			
	BV	Standard	BV	Standard	BV	Standard	BV	Standard	BV	Standard	BV	Standard
	Min		Max		Min		Max		Min		Max	
ROI 1	4.41	55.83	9.60	69.78	1.21	55.90	3.48	65.56	0.41	56.67	2.58	64.60
ROI 2	3.15	51.75	8.87	68.07	1.26	52.12	3.76	66.03	0.62	53.70	2.70	64.10
ROI 3	1.08	12.87	17.69	78.01	0.20	13.23	7.64	73.64	0.13	13.95	6.19	72.58
ROI 4	6.91	56.14	31.79	201.39	2.31	56.54	20.17	200.45	1.41	56.74	17.54	199.04
ROI 5	0.84	15.24	27.21	128.84	0.45	15.55	18.41	127.82	0.30	15.92	14.63	125.15

potentially more affected by noise. Conversely, the greater the radius size, the higher the noise reduction but the less the geometrical details that are retained.

As the radius size increases, the greater the number of incomplete windows that will be processed at specific image locations; e.g., this is indeed the case whenever the MF window approaches the image edges or corners, where the number of window's missing values is maximum. The equation $\frac{100(AB-2r(A+B-2r))}{AB} \geq p$ gives an upper bound for the percentage p of incomplete windows; this bound depends on the image dimensions AxB but also on the windows radius r and can be used to reach a desired percentage of complete windows. For example, if a filter of radius $r = 5$ is applied to the whole test image in Fig. 3, a 95.78% of complete windows are guaranteed; this percentage decreases to 83.70% when $r = 20$ and to 61.97% when $r = 50$. In this way, not only the radius size but a trade-off between the radius size, the image size, the desired geometric details of the output image and the aimed processing time must be taken into account when the performance of a MF algorithm is evaluated.

Figure 4 exhibits the *column-average bin-displacements* needed to update the median (vertical axis) versus the column number (horizontal axis) for each ROI. The former metric can be thought as an experimental realization of the TE ad hoc theoretical metric, measuring the effort in transition between consecutive medians as a column-average. Note that for a fixed radius r the same performance in terms of this metric is shared by any of the standard versions (*H_alg* and *P&H_alg*; dotted line) and by any of the BV-based versions (*BV-based H_alg* and *BV-based P&H_alg*; solid line).

Blue, green and red curves, respectively, describe the effort when radius sizes $r = 1$, $r = 3$ and $r = 5$ are used. Table 2, in turn, shows the minimum and maximum column-average bin-displacements from Fig. 4. Several inferences can be drawn:

- For each ROI, when the radius increases the curves corresponding to the BV-based median computation stabilize in comparison to their standard analogues. This is caused by the greater number of processed values in the MF window that allows the median's optimal BV to come into play; the median is thus quickly updated, and the effort to achieve tends to be zero.
- The curves corresponding to the MF (by using both the standard and the BV-based median computation) of ROI-1 and ROI-2 in Fig. 4 exhibit a rather similar pattern. This might be caused by the shared pattern of seabed reverberation and high variability of pixel values.
- ROI-3 exhibits a higher proportion of acoustic shadow (equivalently, most near zero pixel values): the curves for the standard version indicate that whenever the MF window processes the shadow zone, the column-average effort to update the median decreases (Table 2 suggests that at least 13 fewer displacements are used by the BV-based version in this case, whatever the radius size). Toward the end of the curve the average effort increases again, because this region includes acoustic highlight. As shown in Table 2, a maximum of about 200 displacements are needed in any case. Finally, the average effort needed by the BV-based version tends to stabilize around zero as the radius size increases; this pattern is seen for every ROI.
- ROI-4 includes a significant amount of acoustic highlight, caused by the reflective object (with pixel values near 255). The standard median computation curves in Fig. 4 indicate a higher column-average effort at the beginning of the region, reaching a local maximum of approx. One hundred and seventy-eight displacements to update the median. Next, the curves drops as the presence of seabed reverberation increases: a minimum of approx. Fifty-six displacements is reached for every radius size. Finally, the standard curves climb again until a maximum of 200 displacements is reached whatever the size, which seems to be caused by the increasing amount of acoustic highlight. For the BV-based version, all curves remain stable near zero but become more plain (or constant) as the radius increases.
- When ROI-5 is analyzed, the standard version curves indicate that a significant amount of acoustic shadow (or zero pixel values) is found in this region: these curves

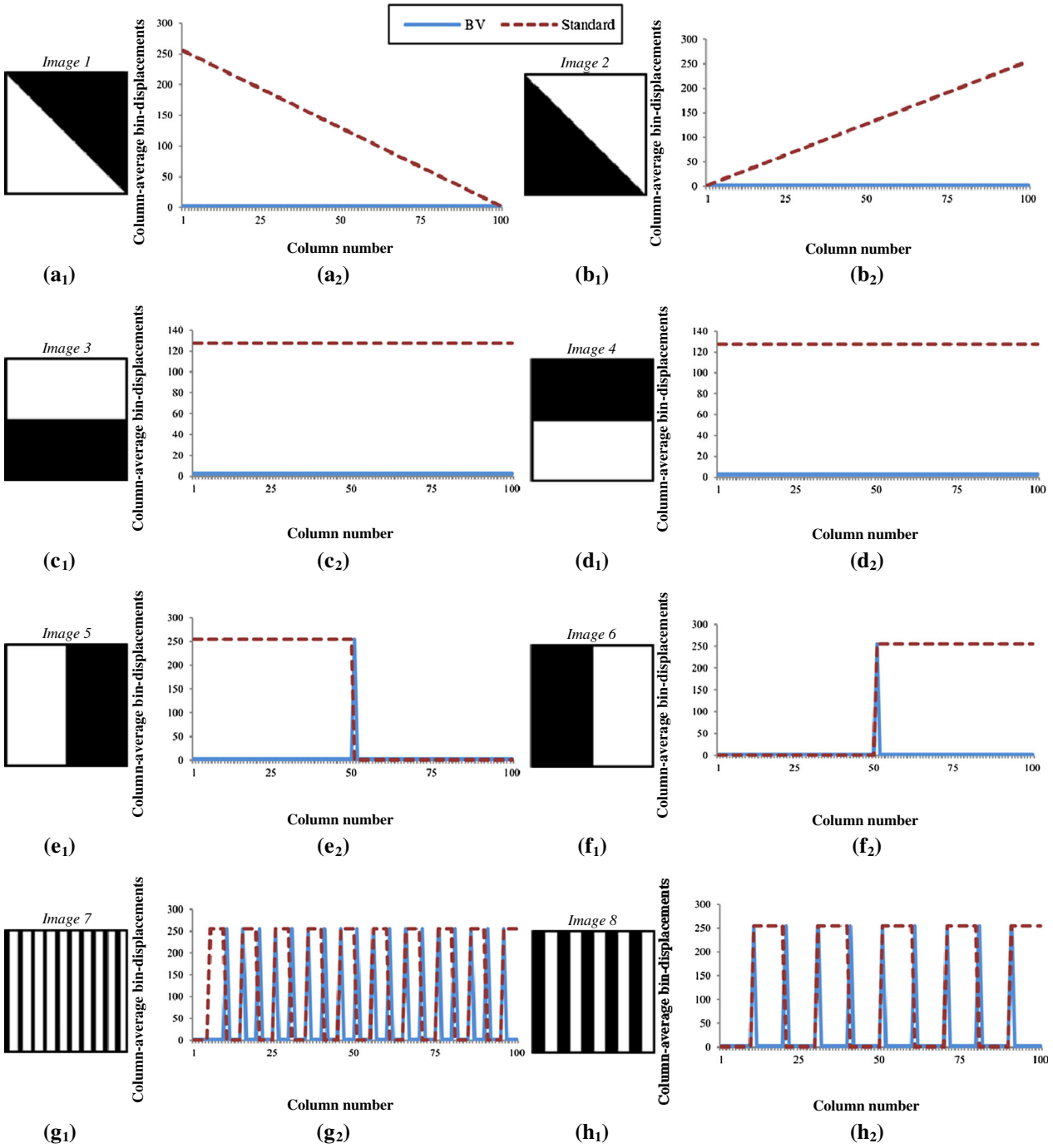

Fig. 5 *Column-average bin-displacements* versus *column number* curves for different synthetic images when a MF of radius size $r = 1$ is used

drop to a minimum of 15 displacements for all three radius sizes. Approaching the right border, the amount of seabed reverberation increases and a small reflective object appears: a maximum of 127 displacements approx. is thus reached for all three radius sizes. The corresponding curves for the BV-based version remain stable near zero most of the time but as the variability of pixel values increases on the right, a maximum of 27 displacements are needed for the radius size $r = 1$; this is caused by a high contrast between light gray levels and dark gry levels (or vice versa), which makes the median to suddenly change.

Figure 5 shows a set of synthetic images designed to illustrate the advantages of BV-based version. All images are of size 100×100 and only the extreme levels of the grayscale 0 (black) and 255 (white) were used. Several inferences can be drawn:

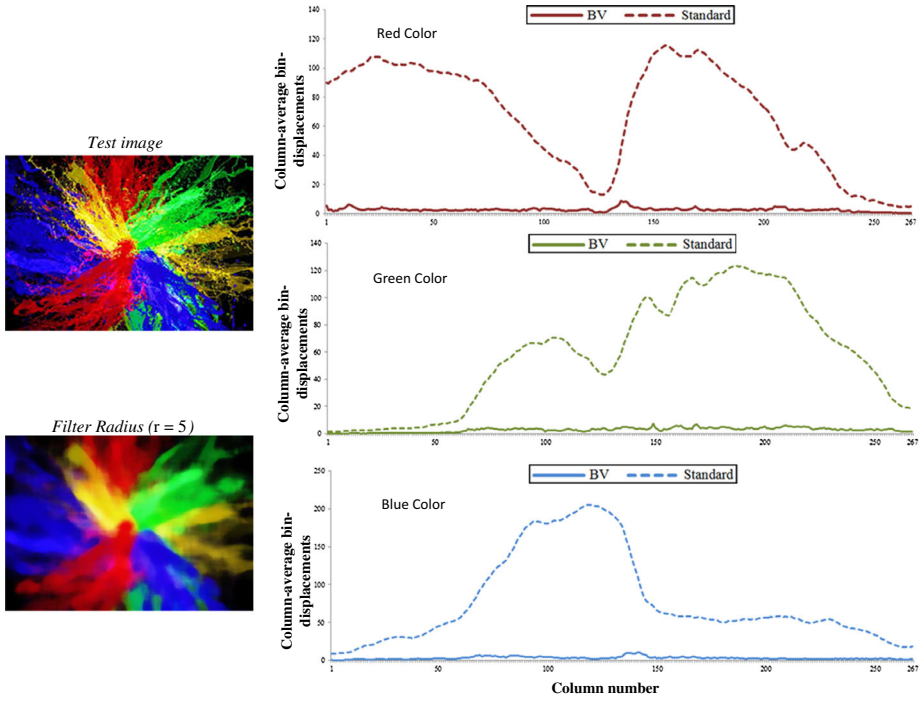

Fig. 6 *Column-average bin-displacements* versus *column number* curves for the RGB image when a MF with radius size $r = 5$ is applied

- Figure 5 (a_1) shows a diagonal pattern of color change. Assuming that MF is performed following a top-down and left-right pattern as usual, the first column has all white pixels (255 values). As the MF window shifts to its right, the frequency of 255 pixel values decreases while the frequency of 0 (black) values increases until a last column full of black pixels is reached. This feature can be perceived in the curve of the standard median computation (Fig. 5 (a_2)), where the column-average bin-displacements required to update the median starts at 255 and descends to 0. Because the search in kernel histogram begins at the 0 level and upward, an average effort of 255 displacements is required to update the median at the beginning; the last column only includes 0 values and no bin-displacement is finally required in average. Alternatively, the curve of the displacement dynamics for the BV-based version indicates that for every column only one maximum change (when switching from 255 to 0) is needed to update the median. Figure 5 (b_1) and (b_2), in turn, exhibits the corresponding pattern for the symmetric case.
- Figure 5 (c1) the pattern of gray-scale variation is now vertical, considering the same two extreme levels: the first 50 rows are completely white, and the remaining 50 rows are completely black. The dynamics of the standard version described in Fig. 5 (c2) shows a column-average effort of 127 bin-displacements to update the median. On the other hand, the BV-based version keeps using only one displacement by column. Figure 5 (d1) and (d2) shows the symmetric case.
- Figure 5 (e_1), in turn, shows a horizontal variation pattern: the first 50 columns are completely white while the last 50 columns are completely black. The dynamics of the standard version described in Fig. 5 (e_2) stabilize around 255 bin-displacements by column to update the median for the first 50 white columns. For the next 50 black columns, a constant average of zero effort is needed to update the median. On the other hand, the BV-based version reaches a peak of 255 average bin-displacements only in column 50 because all windows centered there include approximately the same number of black and white pixels; the effort is due to the transition from 0 to 255 in the grayscale. For the remaining black columns, no displacement in average is needed. Figure 5 (f_1) and (f_2) exhibits the dynamic for the symmetric image.
- Figure 5 (g_1) and (h_1) exhibits a rather similar pattern of variation as those of Fig. 5 (e_1) and (f_1), but with a higher frequency of black and white columns interspersed. Figure 5 (g_1) has a frequency of 5 consecutive columns of the same color, while this frequency is 10 in Fig. 5 (h_1). Analyzing the dynamic curves (Fig. 5 (g_2)), the standard version keeps an average of 255 bin-displacements to update the median for all white columns; for all the black ones, no displacement in average is needed. Instead, the BV-based version reaches peaks of 255 average displacements every 5 columns. A similar pattern applies to Fig. 5 (g_2) but every 10 columns in this case.

Finally, Fig. 6 shows an RGB image with three 8 bit-depth channels ($b = 256$) and the result of MF with radius sizes

1 (9 pixels per window), 3 (49 pixels) and 5 (121 pixels); each channel was processed independently. The image size is 189x267 pixels. The comparison between the standard and the BV-based version is shown in Fig. 6 for each channel. As in Fig. 4, a similar pattern of differences between both versions can be perceived.

5 Final Comments

By assuming a clear distinction between the theoretical analysis of efficiency and the comparison of experimental implementations, this article intends to bring a new insight into the MF capabilities. A detailed review of two of the most popular MF algorithms (Huang et al. and Perreault and Hébert) is offered first in order to understand their main features and also to underline their differences. Updated versions of them, based on the optimal BV of the median are then presented and analyzed in detail. A comprehensive framework to assess the efficiency of MF algorithms in general is introduced in the process. Through six different and objective-abstract metrics the theoretical efficiency of the four MF algorithms under consideration is assessed in detail; these metrics could also be used to approach the theoretical analysis of a wider class of algorithms. An additional ad hoc metric, the transition effort rate, provides a rather intuitive explanation of the experimental results: the BV-based versions clearly outperform their corresponding standard versions, even the latter are equally or more expensive than the former in terms of the remaining metrics.

The experimental results combined both real and synthetic data including a RGB example and the experimental computation of the transition effort rate. The differences between the standard and BV-based versions of the analyzed MF algorithms are thus highlighted and objectively quantified. Overall, the BV-based algorithms noticeably outperform their corresponding standard versions either for a single channel or a multichannel image.

Lines of future work might include the use of BV-based MF versions for edge detection in object segmentation, and the analysis of MF algorithms under alternative theoretical models of computation.

References

1. Cormen, T.H., Stein, C., Rivest, R.L., Leiserson, C.E.: Introduction to Algorithms. McGraw-Hill, New York (2001)
2. Skiena, S.S.: The Algorithm Design Manual. Springer, London (2008)
3. Tukey, J.: Exploratory Data Analysis. Addison-Wesley, Menlo Park (1977)
4. Maragos, P., Schafer, R.: Morphological filters—part II: their relations to median, order-statistic, and stack filters. IEEE Trans. Acoust. Speech Signal Process. **35**, 1170–1184 (1987)
5. Pratt, W.K.: Digital Image Processing: PIKS Inside. Wiley, New York (2001)
6. Eng, How-Lung, Ma, Kai-Kuang: Noise adaptive soft-switching median filter. IEEE Trans. Image Process. **10**, 242–251 (2001)
7. Nelson, T.R., Pretorius, D.H.: Three-dimensional ultrasound of fetal surface features. Ultrasound Obstet. Gynecol. **2**, 166–174 (1992)
8. Carayon, P., Portier, M., Dussossoy, D., Bord, A., Petitpretre, G., Canat, X., Le Fur, G., Casellas, P.: Involvement of peripheral benzodiazepine receptors in the protection of hematopoietic cells against oxygen radical damage. Blood **87**, 3170–3178 (1996)
9. LeBas, T.P., Mason, D.C., Millard, N.C.: TOBI image processing: the state of the art. J. Ocean. Eng. **20**, 85–93 (1995)
10. Villar, S.A., Torcida, S., Acosta, G.G.: Un Enfoque Novedoso del Filtro de Mediana para el Suavizado de Señales Acústicas de Sonar de Barrido Lateral. Presented at the ARGENCON 2014, San Carlos de Bariloche-Neuquen-Argentina June 11 (2014)
11. Chen, Zhao-Yan, Wang, Tong, Ma, Nan: Accurate baseline estimation for synthetic aperture radar-ground moving target indication systems based on co-registration and median filtering. Radar Sonar Navig. IET. **8**, 607–615 (2014)
12. Juhola, M., Katajainen, J., aRaita, T.: Comparison of algorithms for standard median filtering. IEEE Trans. Signal Process. **39**, 204–208 (1991)
13. Langsam, Y., Augenstein, J., Tenenbaum, A.: Data Structures Using C and C++. Prentice Hall, Englewood Cliffs (1995)
14. Corwin, E., Logar, A.: Sorting in linear time-variations on the bucket sort. J. Comput. Sci. Coll. **20**, 197–202 (2004)
15. Thomas, C.H., Charles, L.E., Ronald, R.L., Clifford, S.: Introduction to Algorithms. MIT Press and McGraw-Hill, Cambridge (2001)
16. Tibshirani, R.J.: Fast Computation of the Median by Successive Binning (2008). ArXiv Prepr. arXiv:0806.3301
17. Suomela, J.: Median Filtering is Equivalent to Sorting (2014). ArXiv Prepr. arXiv:1406.1717
18. Huang, T., Yang, G., Tang, G.: A fast two-dimensional median filtering algorithm. IEEE Trans. Acoust. Speech Signal Process. **27**, 13–18 (1979)
19. Weiss, B.: Fast median and bilateral filtering. ACM Trans. Graph. **25**, 519–526 (2006)
20. Gil, J., Werman, M.: Computing 2-D min, median, and maxfilters. IEEE Trans. Pattern Anal. Mach. Intell. **15**, 504–507 (1993)
21. Perreault, S., Hebert, P.: Median filtering in constant time. IEEE Trans. Image Process. **16**, 2389–2394 (2007)
22. Urbach, E.R., Wilkinson, M.H.F.: Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. IEEE Trans. Image Process. **17**, 1–8 (2008)
23. Alekseychuk, A.: Hierarchical recursive running median. In: 19th IEEE International Conference on Image Processing (ICIP 2012), Lake Buena Vista, Orlando, FL, USA, September 30– October 3, 2012, pp. 109–112 (2012)
24. Huber, P.J.: Robust Statistics. Wiley, New York (1981)
25. Kluge, W.: Abstract Computing Machines: A Lambda Calculus Perspective. Springer, Berlin (2005)
26. Diehl, S., Hartel, P., Sestoft, P.: Abstract machines for programming language implementation. Future Gener. Comput. Syst. **16**, 739–751 (2000)
27. Jaime, F.J., Hormigo, J., Villalba, J., Zapata, E.L.: New SIMD instructions set for image processing applications enhancement. In: 15th IEEE International Conference on Image Processing, 2008 (ICIP 2008), pp. 1396–1399 (2008)
28. Alparone, L., Cappellini, V., Garzelli, A.: A coarse-to-fine algorithm for fast median filtering of image data with a huge number of levels. Signal Process. **39**, 33–41 (1994)

29. Acosta, G.G., Villar, S.A.: Accumulated CA-CFAR process in 2-D for online object detection from sidescan sonar data. IEEE J. Ocean. Eng. **40**, 558–569 (2015)
30. Villar, S.A., Acosta, G.G., Sousa, A.L., Rozenfeld, A.: Evaluation of an efficient approach for target tracking from acoustic imagery for the perception system of an autonomous underwater vehicle. J. Adv. Robot. Syst. InTech. **11**, 1–13 (2014)

Sebastián A. Villar graduated as Systems Engineer at the Facultad de Ciencias Exactas from the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Argentina (2009), and as Masters in Business Administration (MBA) at the Facultad de Ciencias Economicas from UNCPBA (2011), and as Ph.D. in Engineering, at Facultad de Ingeniería of UNCPBA (2014). He is also a researcher of the Argentinean National Research Council (CONICET) with a scholarship, working in Engineering Group INTELYMEC (Av. del Valle 5737-B7400JWI Olavarría; Argentina), UNCPBA.

Sebastián Torcida graduated as a bachelor in Mathematics (1995) and as a Masters in Mathematics (2001), both from the Mathematics Department at the Facultad de Ciencias Exactas from the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Argentina. He is a full time researcher in applied mathematics and multivariate statistics, collaborating with colleagues from different disciplines.

Gerardo G. Acosta graduated as Engineer in Electronics at the Universidad Nacional de La Plata, Argentina (1988), and as Ph.D. in Computer Science, at the Universidad de Valladolid, Spain (1995). He is currently a Full Professor in Control Systems (Electronic Area) at the Facultad de Ingenieria from the Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA), Argentina. He is a researcher of the Argentinean National Research Council (CONICET) since 1997, and the head of the research group "INTELYMEC", CIFICEN, CONICET-UNCPBA. He is a Senior Member of the IEEE since 2001, Chairman of the Argentinean Chapter of the IEEE Computational Intelligence Society (2007–2008), and received the 2010 Outstanding Chapter Award from CIS. In addition, he is the current Chairman of the Argentinean Chapter of the IEEE Oceanic Engineering Society being one of its funders. He has been the leader of more than ten research projects funded by the argentinean government, the spanish government and the european union. He has been invited as a professor of Ph.D programs in Argentina and Spain, he is the current Director of the PhD program at the Facultad de Ingeniería-UNCPBA, and serves as a reviewer and a member of the scientific committee of several national and international journals and conferences.